Improving Model Performance, Portability and Productivity with Apache TVM and the OctoML platform

ModSim'21

Luis Ceze

Co-founder & CEO, OctoML.

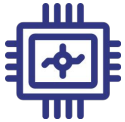
Professor, University of Washington.

Credit goes to many collaborators at OctoML, UW, the Apache TVM community!

The Problem: Machine Learning is hard and costly to deploy


Engineering

Cloud costs

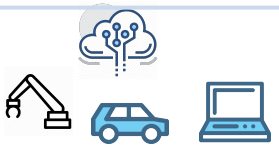


Silicon resources

Data Collection and Processing

Model Development & Training

Model Optimization & Deployment



Cloud, Mobile, Edge Inference

Large gap not covered by typical “MLops”

Performance/efficiency gap

Model optimization and tuning is essential for viable deployment.

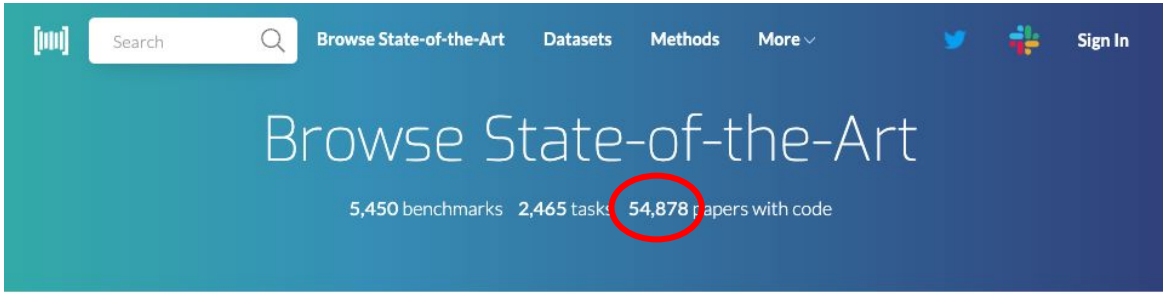
Productivity gap

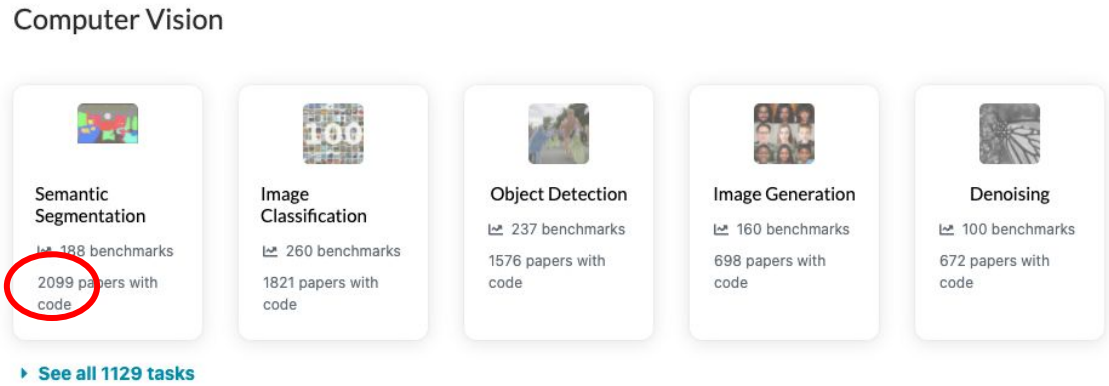
Weeks to months of effort to get a model ready for deployment.

Portability gap

Changing deployment HW requires significant manual effort. Vendor lock-in.

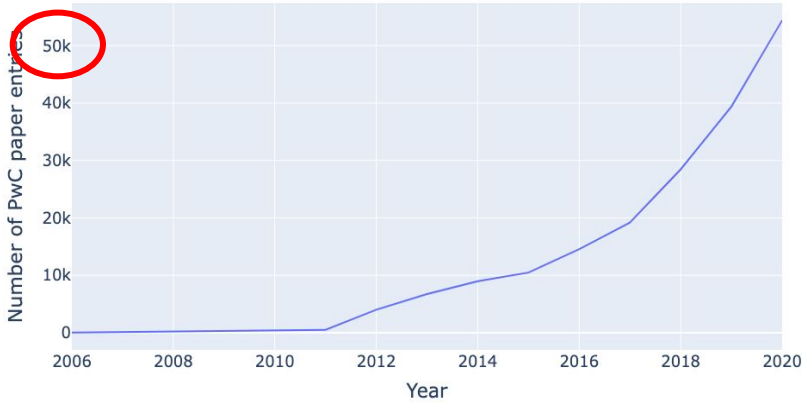
Trend: Machine learning workload diversity is exploding

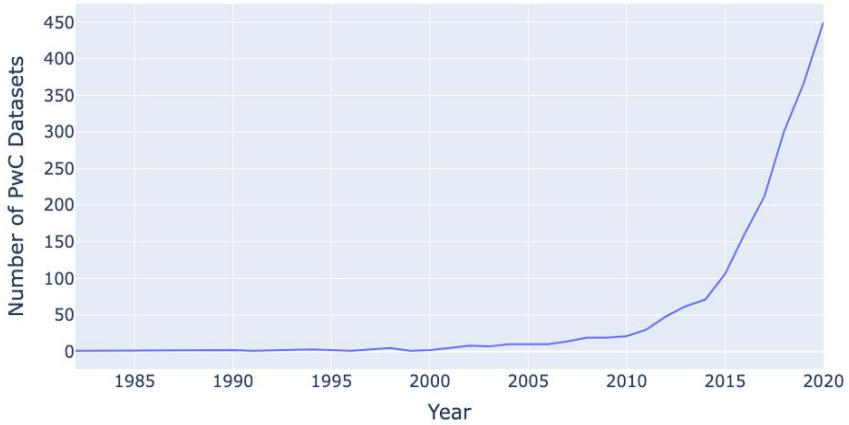




Trend: Machine learning workload diversity is exploding

paperswithcode.com






```
In [76]: len(abstracts)
Out[76]: 230764
```

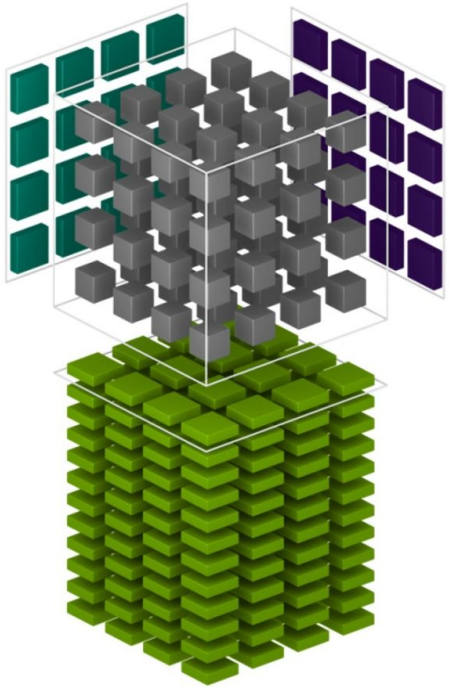
* Non-cumulative plots

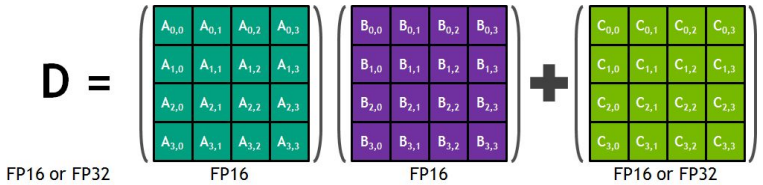
OctoML

Trend: ML hardware capabilities exploding

E.g., TensorCores

- FP16
- Int8
- Int4, Int2, Int1
- Bfloat16, TF32
- FP64
- Tensor Core sparsity

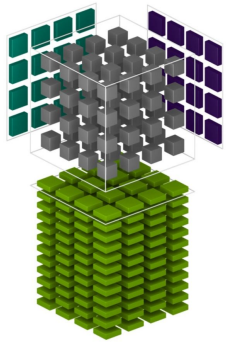


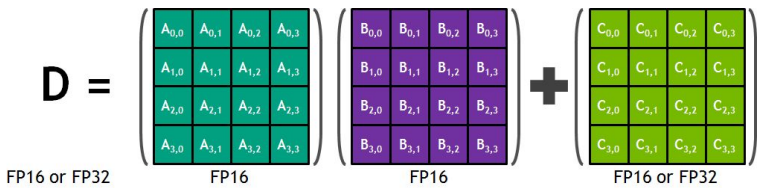


Trend: ML hardware capabilities exploding

More programmability! E.g.,

- Asynchronous copy instruction loads data directly from global memory into shared memory, optionally bypassing L1 cache
- New instructions for L2 cache management and residency controls.
- New warp-level reduction instructions supported by CUDA Cooperative Groups.

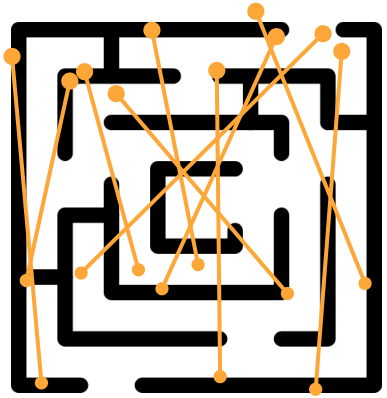




An exploding ecosystem...

Rapidly evolving ML software ecosystem

- Use-case or HW-specific stack often hand-written
- Painful and unproductive for users
- Unsustainable for HW/platform vendors: need to keep up with model and framework evolution

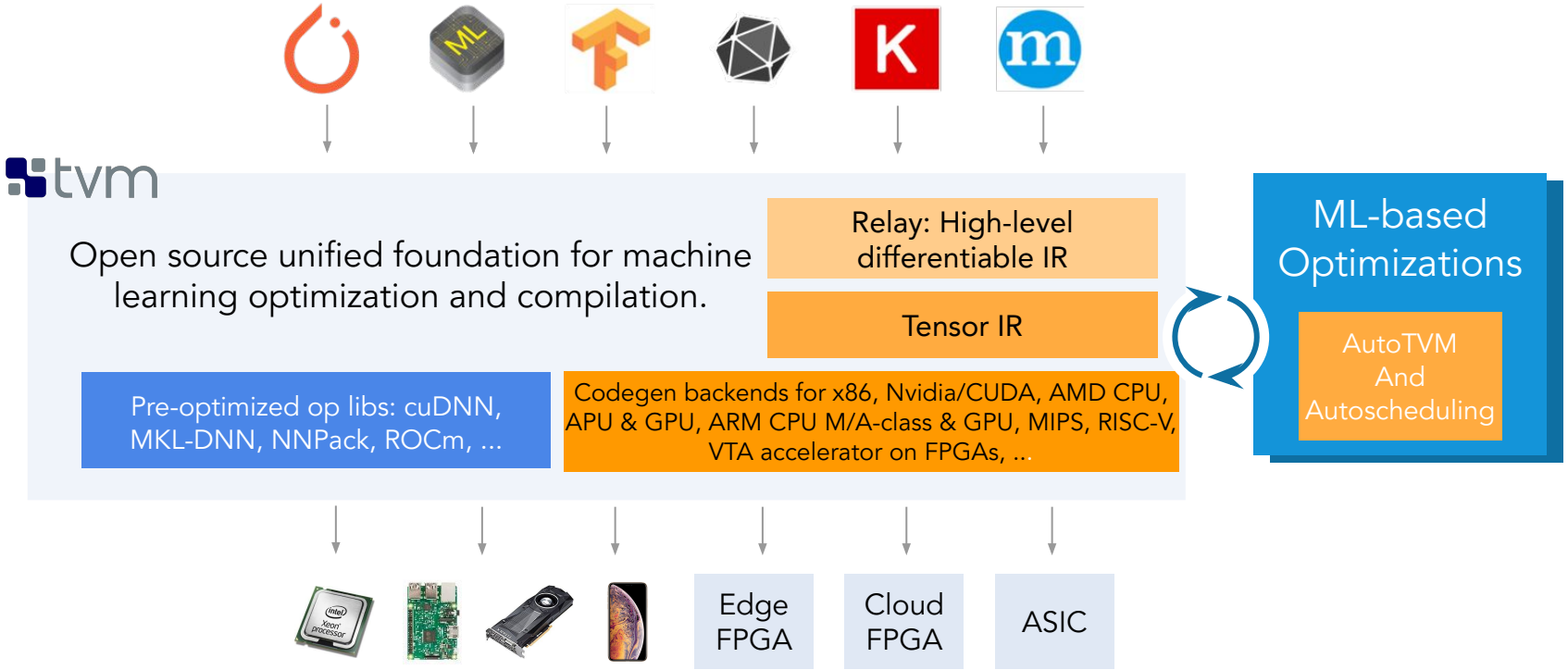


Cambrian explosion of HW backends

Automated, open source, unified optimization and compilation framework for deep learning.

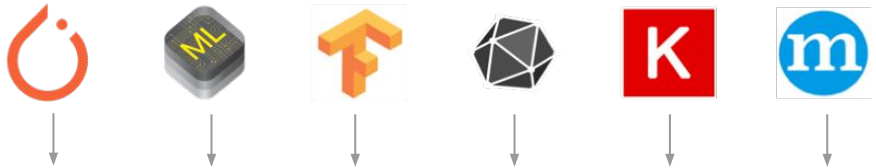
Model In, HW-specific, native code out.

TVM: Automatic ML optimizer, compiler and runtime



TVM: Automatic ML optimizer, compiler and runtime

w/ state-of-the-art performance



Reduce model time-to-market

Build your model once, run anywhere

Cut capital and operational ML costs

tvm

Open source unified foundation for machine learning optimization and compilation.

Relay: High-level differentiable IR

Tensor IR

Pre-optimized op libs: cuDNN, MKL-DNN, NNPack, ROCm, ...

Codegen backends for x86, Nvidia/CUDA, AMD CPU, APU & GPU, ARM CPU M/A-class & GPU, MIPS, RISC-V, VTA accelerator on FPGAs, ...

Edge FPGA

Cloud FPGA

ASIC

OctoML

TVM is an industry standard open source ML stack

Every "Alexa" wake-up today across all devices uses a model optimized with TVM

"[TVM enabled] real-time on mobile CPUs for free...We are excited about the performance TVM achieves." More than 85x speed-up for speech recognition model.

Bing query understanding: 112ms (Tensorflow) -> 34ms (TVM). QnA bot: 73ms->28ms (CPU), 10.1ms->5.5ms (GPU) - TVMconf 2019.

"TVM is key to ML Access on Hexagon" - Jeff Gehlhaar, VP Technology

Unified ML stack for CPU, GPU, NPU built on TVM. TVMconf 2020.

Open source
~570+ contributors from industry and academia.

950+ attendees

13% HW vendors,
35% research, **50%+ ML end user.**

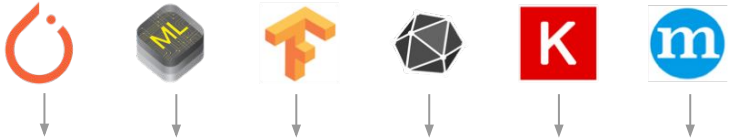
Cross-product of {Models} x {Hardware} is large.
Strong community support makes diversity manageable and ensures future-proofness.

OctoML

Why use Apache TVM?

ML end user: *wants to quickly optimize their model on best/chosen HW target.*

- *ML engineer for production apps*
- *Product R&D*
- *ML researchers*



tvm

Open source, optimization and compilation framework for deep learning.

Backends for x86, nVidia/CUDA, AMD, ARM CPU M/A-class & GPU, MIPS, RISC-V, Accelerators, FPGAs, ...

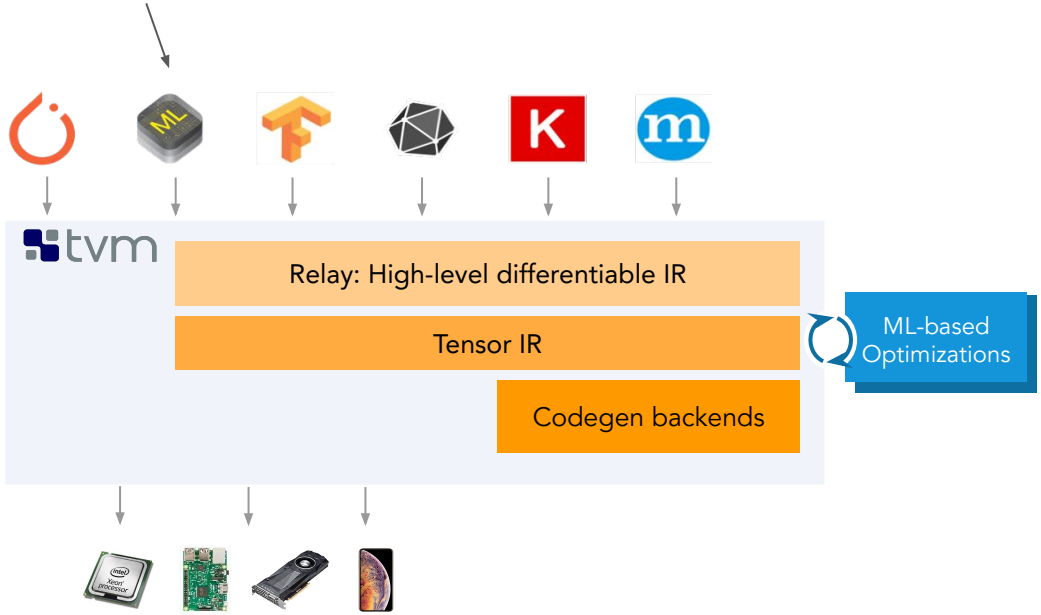
Edge FPGA | Cloud FPGA | ASIC

HW chip vendor & platform provider: *wants to offer their customers the best ML SW stack for all models*

- *System SW engineers*
- *Technical sales demos*

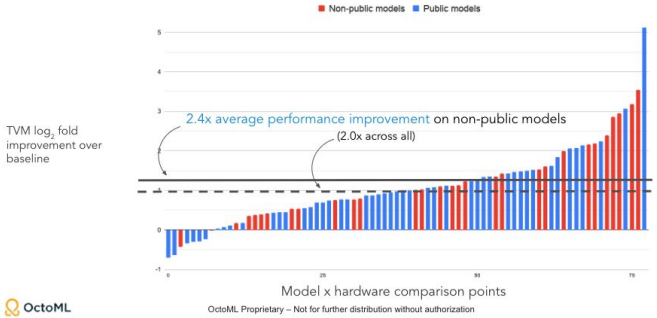
TVM as a compiler and runtime framework

Need: End to end DL model optimization

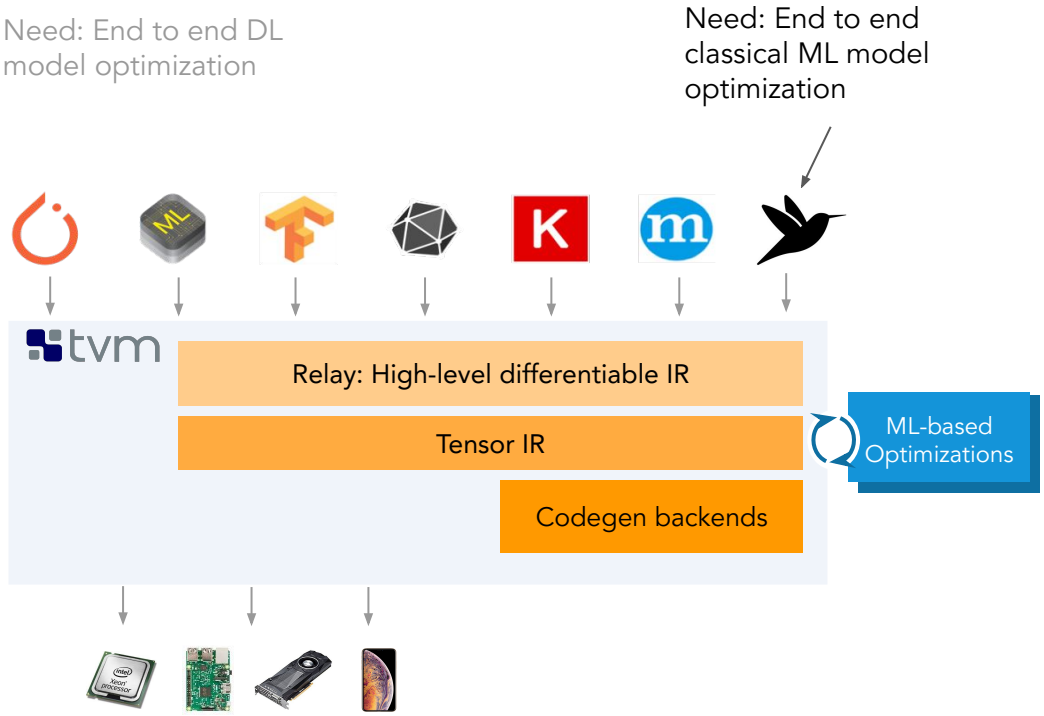


- OctoML historical data
  - 2x avg improvement (up to 30x)

TVM Performance at OctoML



TVM as a compiler and runtime framework



Case study:

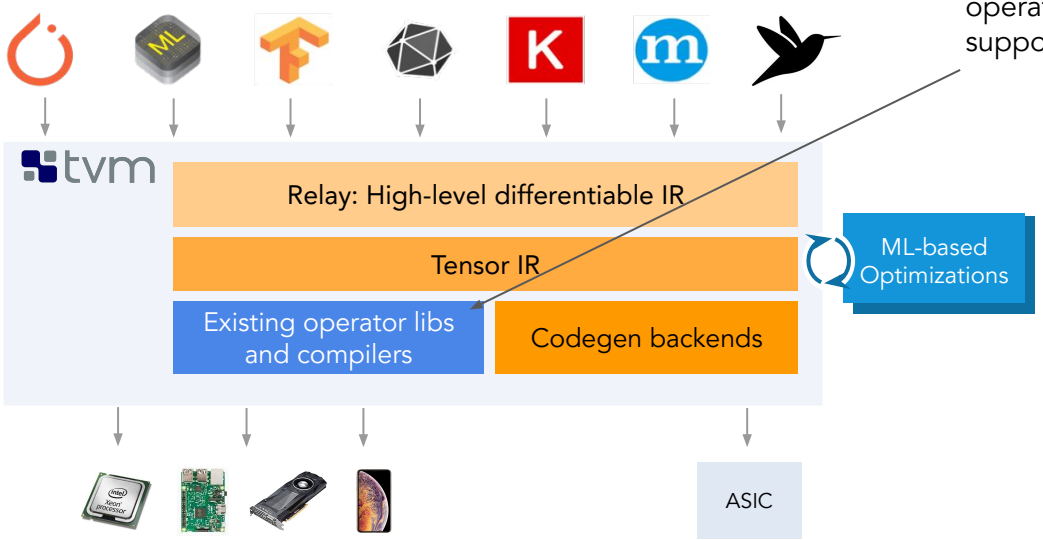
- Support decision tree optimization
  - Convert to sparse tensors, work w/ Microsoft
- Hummingbird
  - Microsoft Azure Data
  - 2-3x throughput increases
  - Broader algorithm support on GPU and future accelerators

TVM as a compiler and runtime framework

Need: End to end DL model optimization

Need: End to end classical ML model optimization

Need: Enabling existing kernel operator libraries to support frameworks.



Case study:

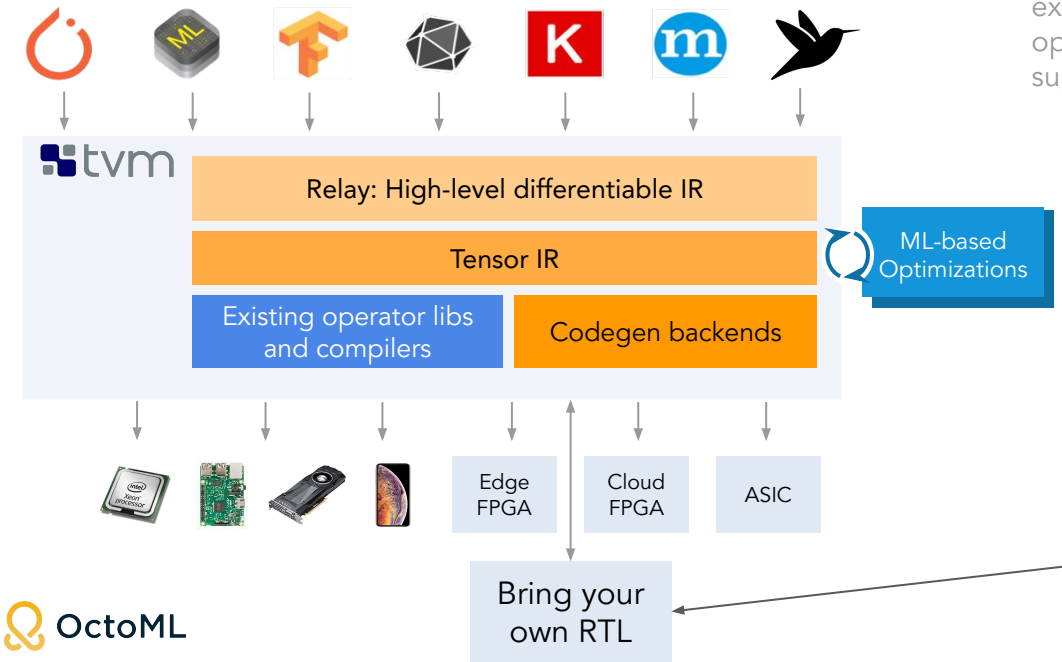
- ARM Ethos NPU
  - Both codegen and existing
  - Enables single compiler stack across CPU, GPU, and NPU
- Amazon Inferentia

TVM as a compiler and runtime framework

Need: End to end DL model optimization

Need: End to end classical ML model optimization

Need: Enabling existing kernel operator libraries to support frameworks.



Case study:
VTA Open Source DL accelerator and "Bring your own RTL"

- Enables rapid feedback loops for SW/HW codesign
- Verilator and other EDA simulators supported

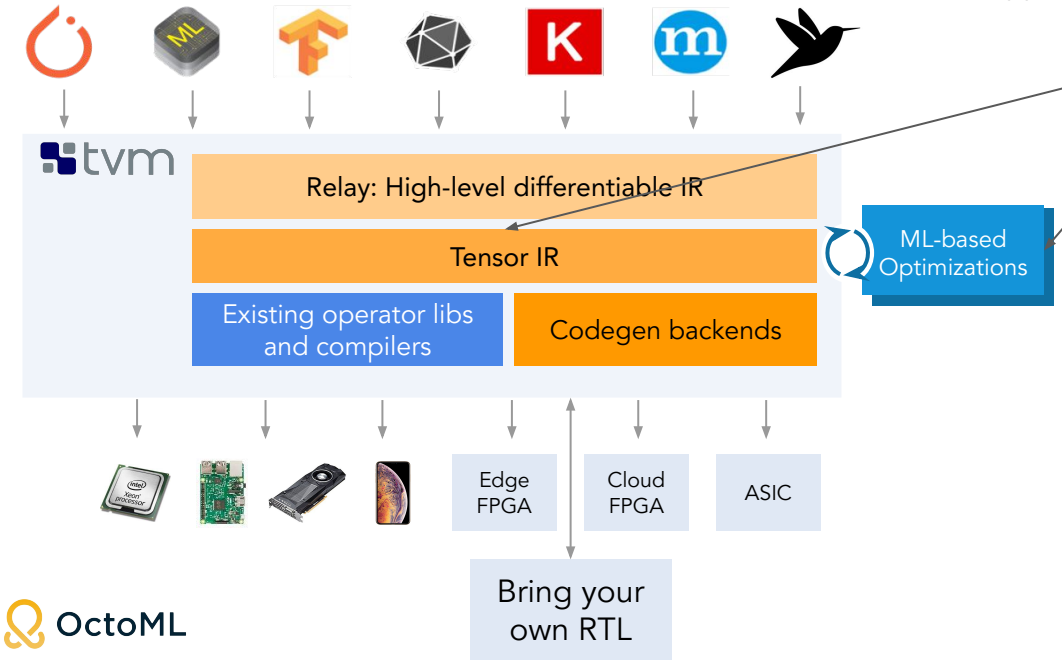
Need: Support rapid HW/SW codesign.

TVM as a compiler and runtime framework

Need: End to end DL model optimization

Need: End to end classical ML model optimization

Need: Enabling existing kernel operator libraries to support frameworks.



Need: Enable users to write custom kernels with:

1. Full control (eg: a "nicer" CUDA)
2. Autotuning
3. Fully automatic autoscheduling

Case study:
Huggingface PruneBERT

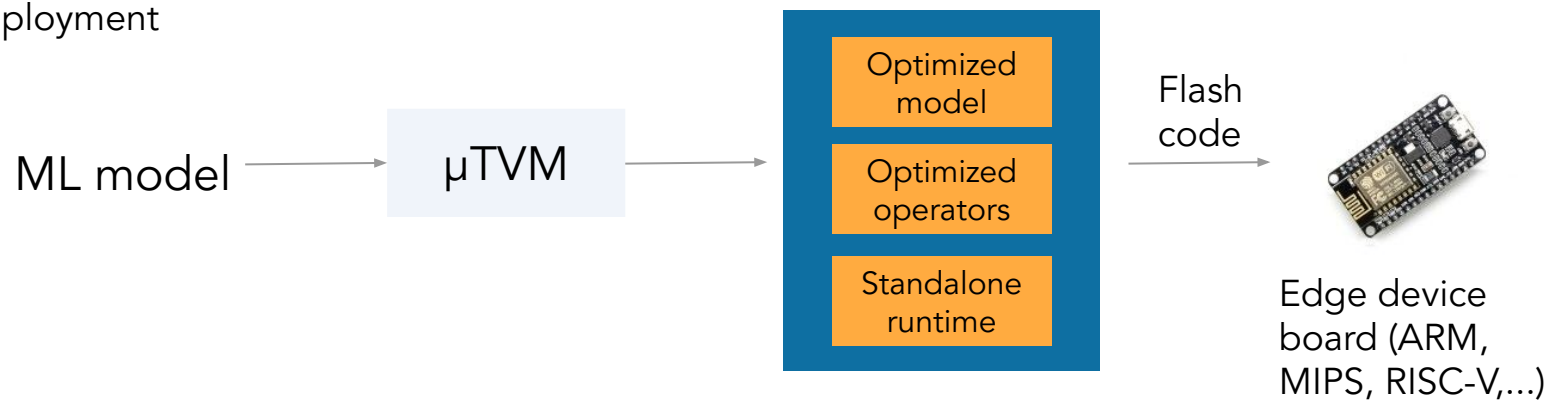
- 2x acceleration on AMD CPUs
- 3x on NVIDIA and AMD GPUs

Kernel wise: 3-10x faster than cuBLAS and cuSPARSE
Single bit end to end acceleration (6-15x)

Need: Support rapid HW/SW codesign.

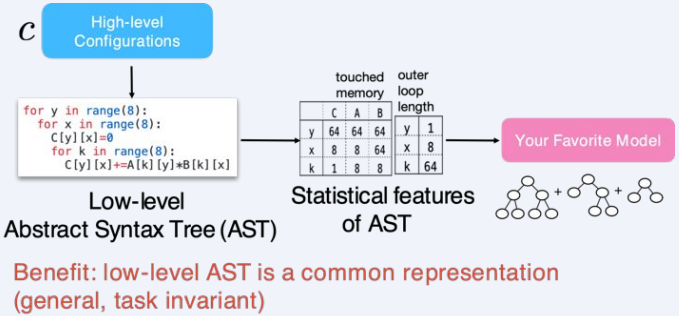
µTVM - Bare-metal model deployment for edge devices

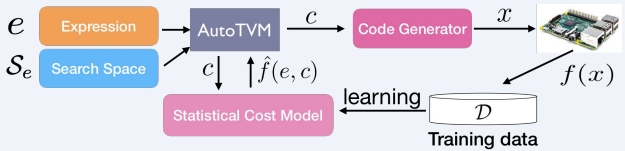
Optimize, compile and package model for standalone bare metal deployment



See recent demo on TVM for Azure Sphere deployment.

ML-based optimizations



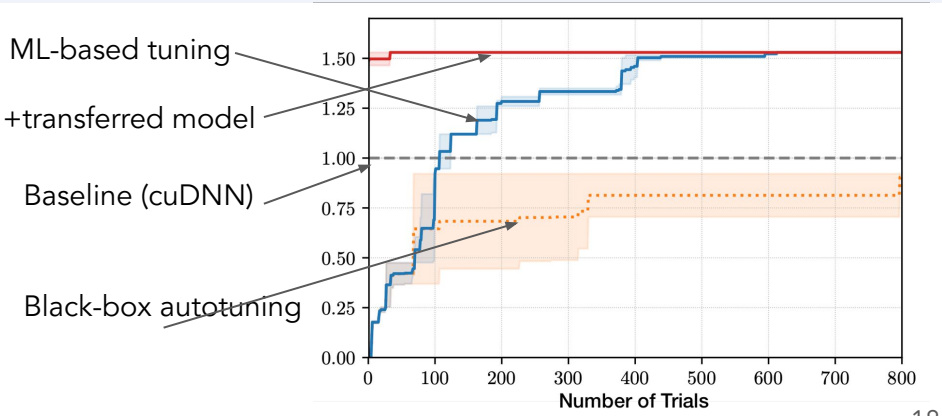


Automatically adapt to hardware type by learning, transferable to other tasks.

Extract hierarchical optimization search space from naive implementation (auto-scheduling).

Start from repository of existing cost models, augment with more experiments (ML-based cost models).

The more it is used, the better it gets!



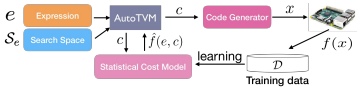
OctoML

TVM technology advantages summary

Broad framework/model importer coverage: Caffe {2}, CoreML, Darknet, Keras, MXNet, ONNX, PyTorch, TensorFlow, TFLite.

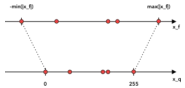
Support for training jobs coming soon.

Clean extensibility to new HW targets.

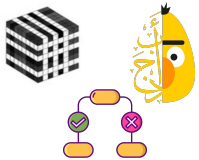


ML-driven auto-tuning/codegen

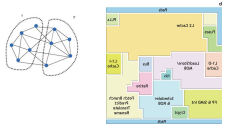
Bare-metal compilation for self-contained deployment in IoT and browsers (WASM/WebGPU)



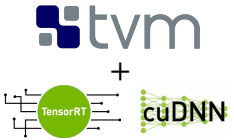
HW-aware quantization



Support for sparse tensors (NLP, classical ML)



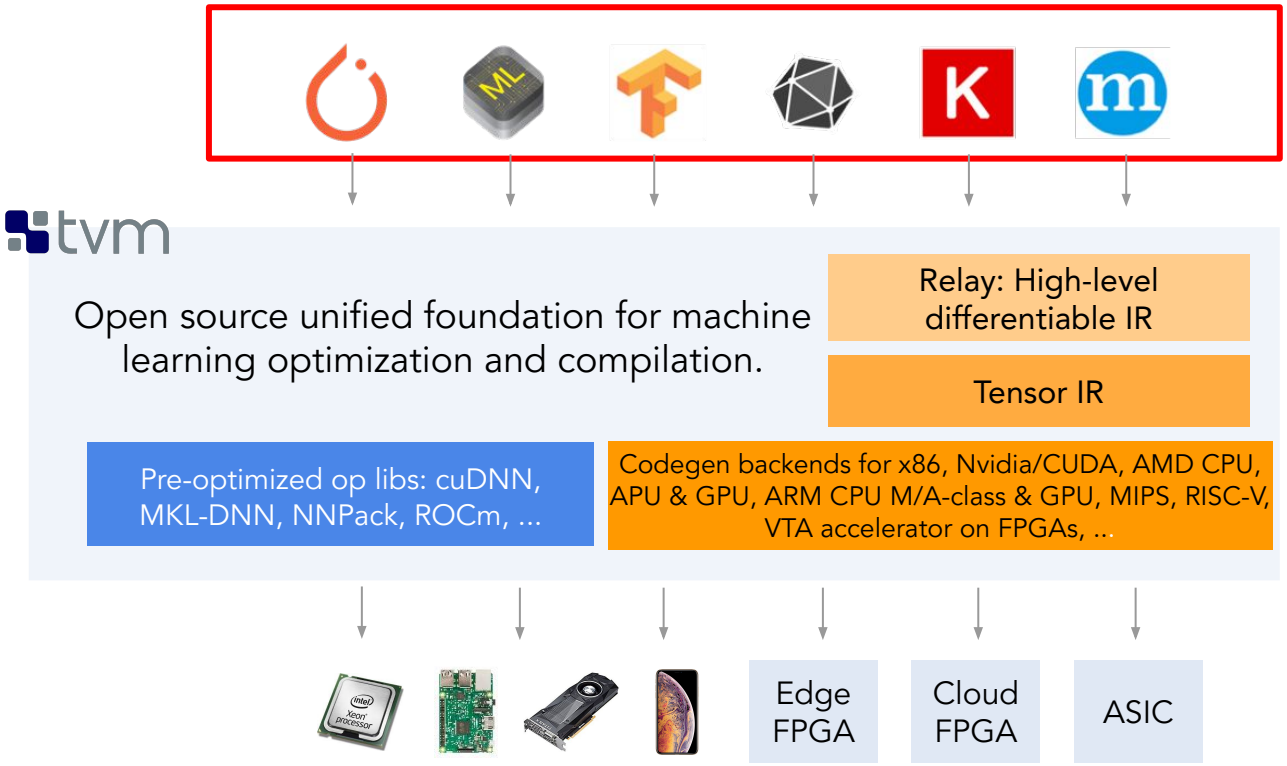
Split-graph execution for heterogeneous HW



Mix and match with operator libs and native compiler stacks

OctoML

TVM deep dive



Importers from a variety of formats into TVM.

- Caffe {2}
- CoreML
- Darknet
- Keras
- MXNet
- ONNX
- PyTorch
- TensorFlow
- TFLite

Shape Information
and Model Definition

```
shape_dict = {“input”: x.shape}
mod, params = relay.frontend.from_onnx(onnx_model, shape_dict)

with tvm.transform.PassContext(opt_level=1):
    intrp = relay.create_executor("graph", mod, tvm.cpu(0), ”llvm”)

tvm_output = intrp.evaluate()(tvm.nd.array(x), **params).asnumpy()
```

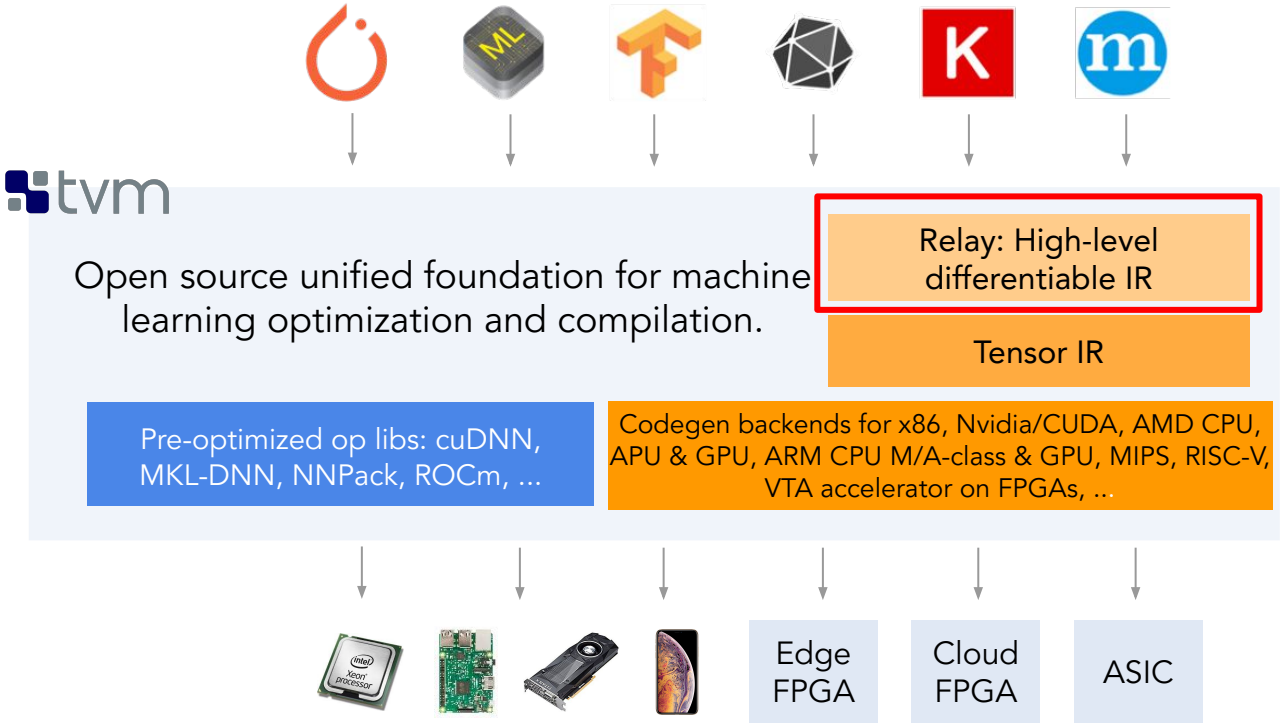
Compile and Create Executor

```
shape_dict = {“input”: x.shape}
mod, params = relay.frontend.from_onnx(onnx_model, shape_dict)

with tvm.transform.PassContext(opt_level=1):
    intrp = relay.create_executor("graph", mod, tvm.cpu(0), ”llvm”)

tvm_output = intrp.evaluate()(tvm.nd.array(x), **params).asnumpy()
```

```
shape_dict = {“input”: x.shape}
mod, params = relay.frontend.from_onnx(onnx_model, shape_dict)

with tvm.transform.PassContext(opt_level=1):
    intrp = relay.create_executor("graph", mod, tvm.cpu(0), ”llvm”)

tvm_output = intrp.evaluate()(tvm.nd.array(x), **params).asnumpy()
```

Run →

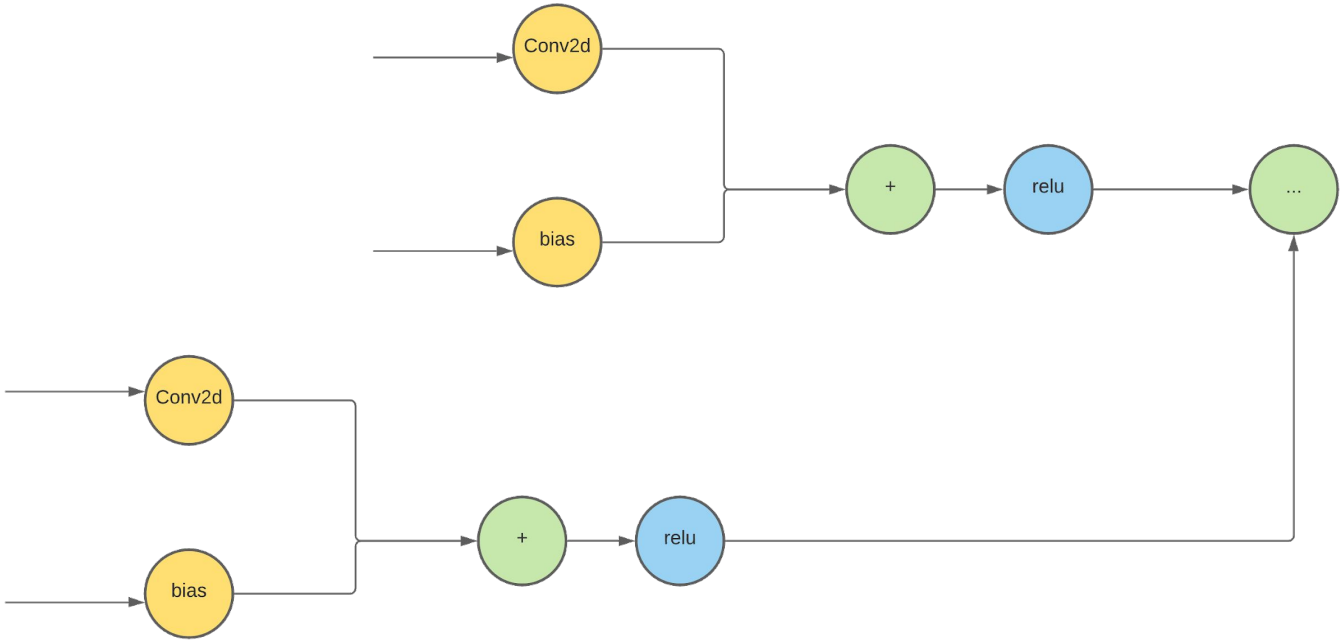
TVM deep dive



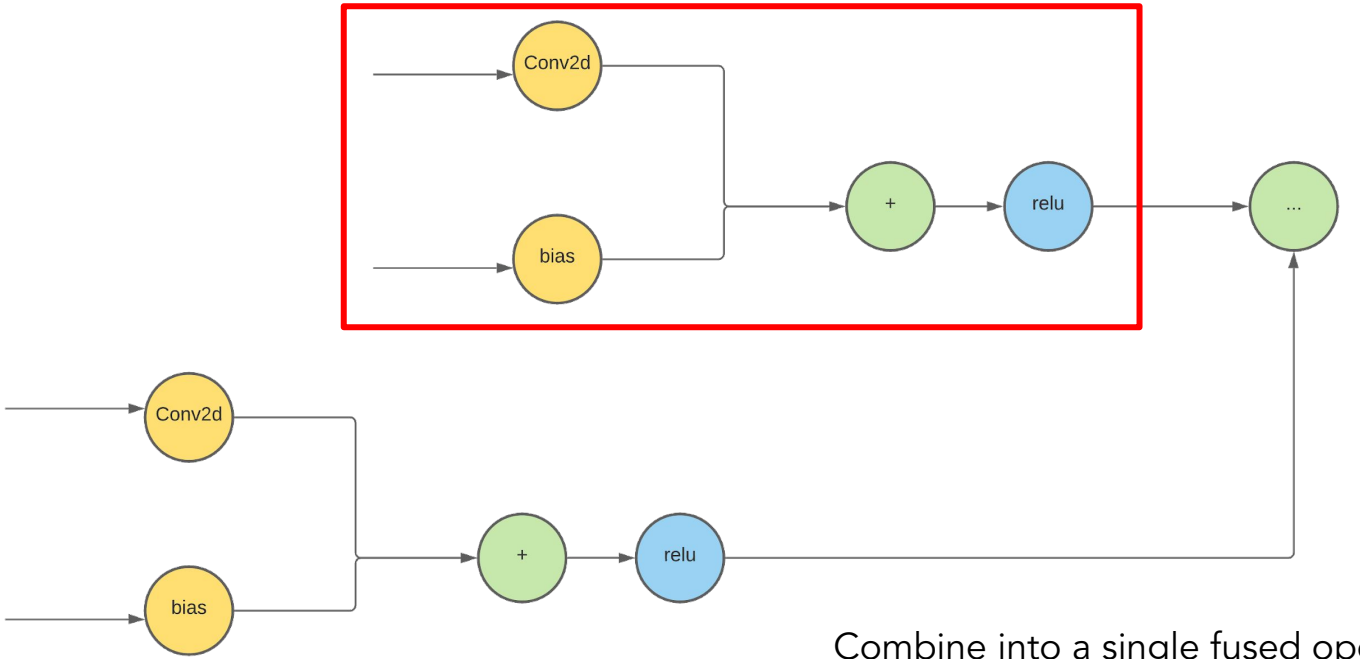
A functional graph-level IR, supports richer programming model.

Functions, data types, primitive operations, tensors, and more.

Relay

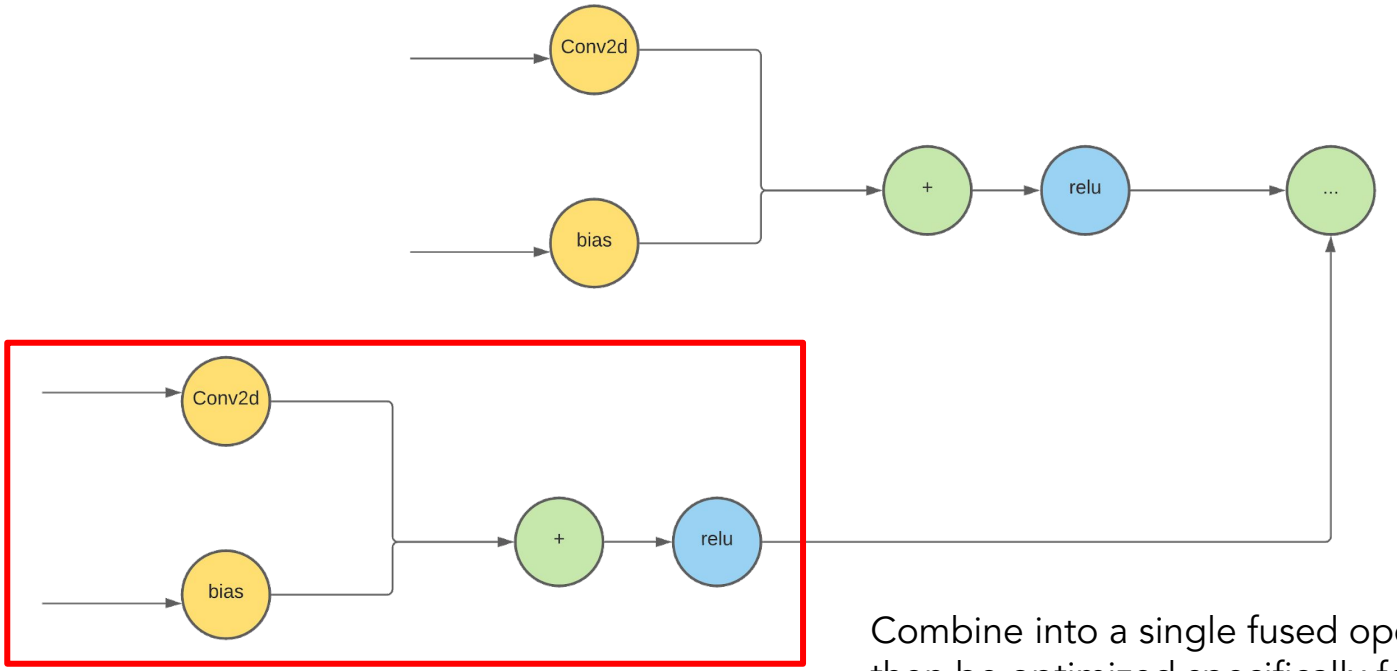


Relay: Fusion



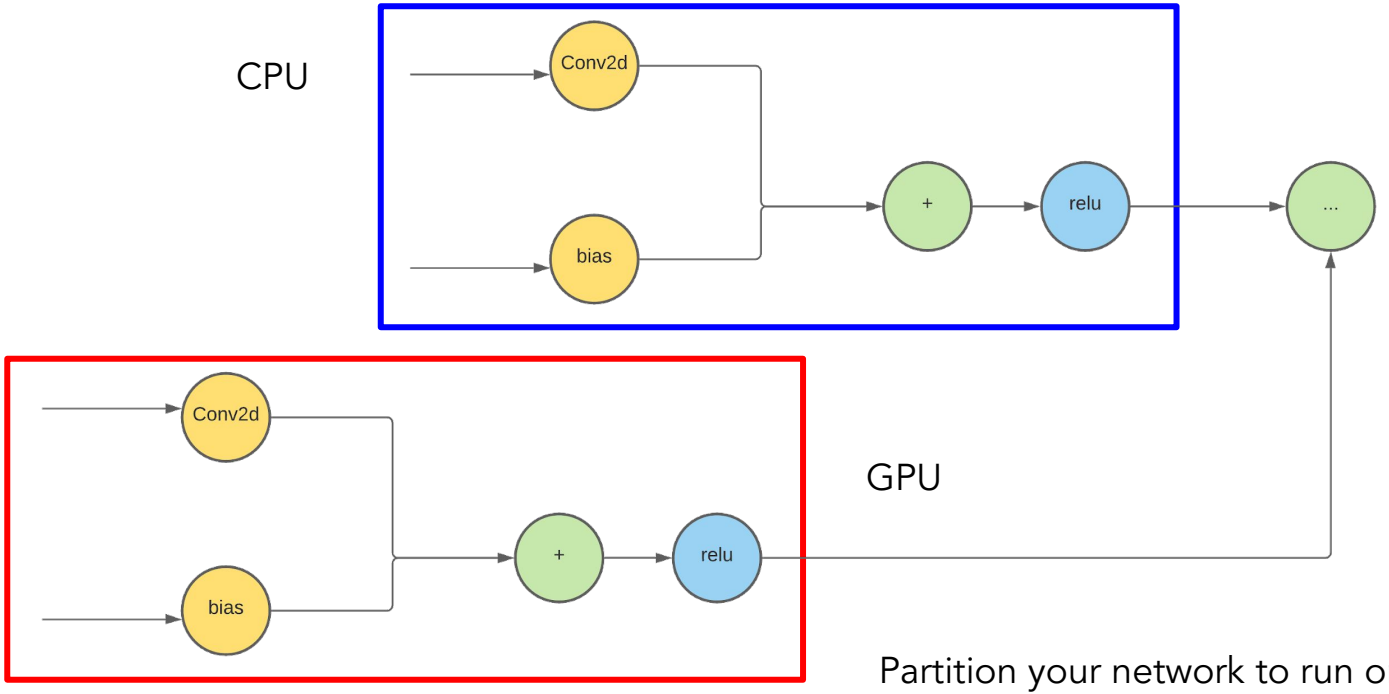
Combine into a single fused operation which can then be optimized specifically for your target.

Relay: Fusion



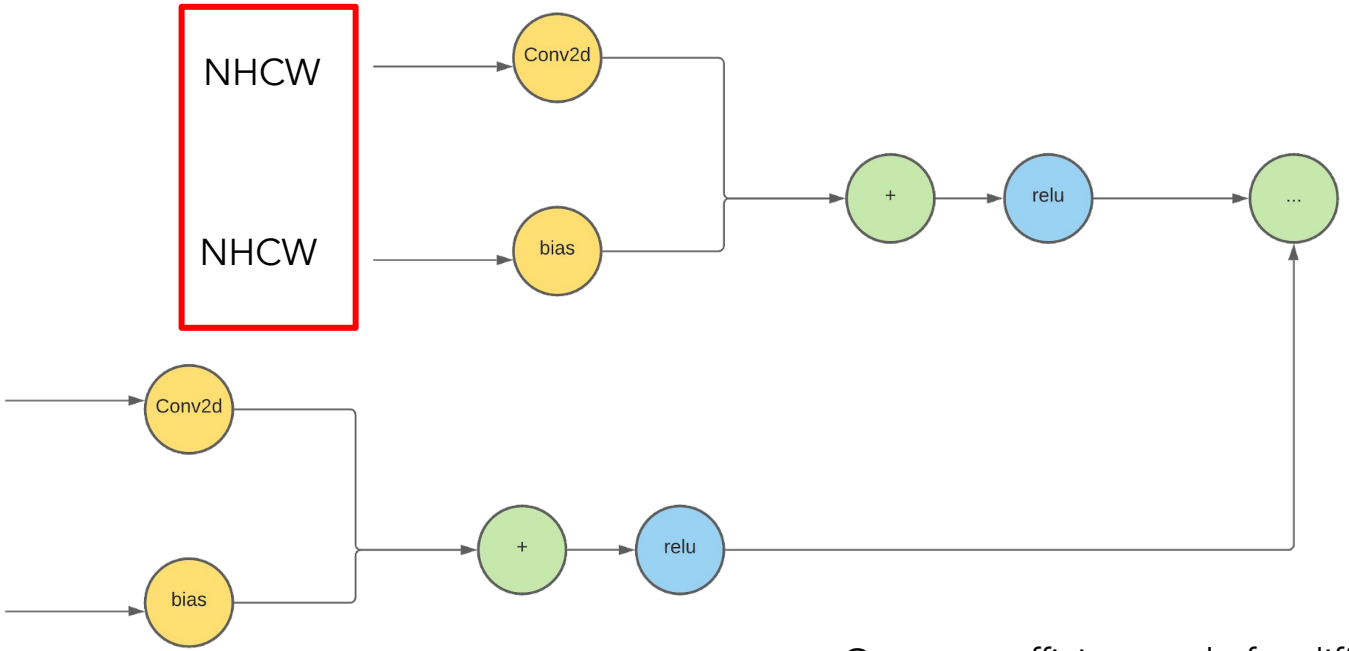
Combine into a single fused operation which can then be optimized specifically for your target.

Relay: Device Placement



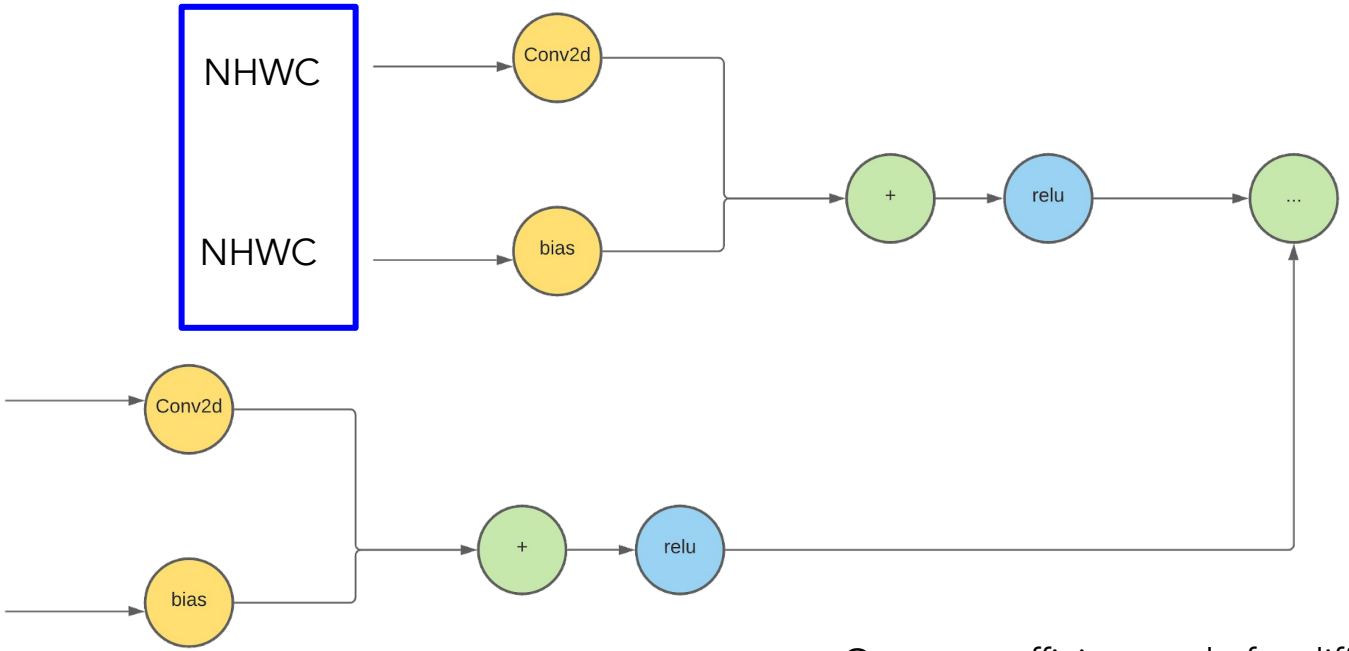
Partition your network to run on multiple devices.

Relay: Layout Transformation



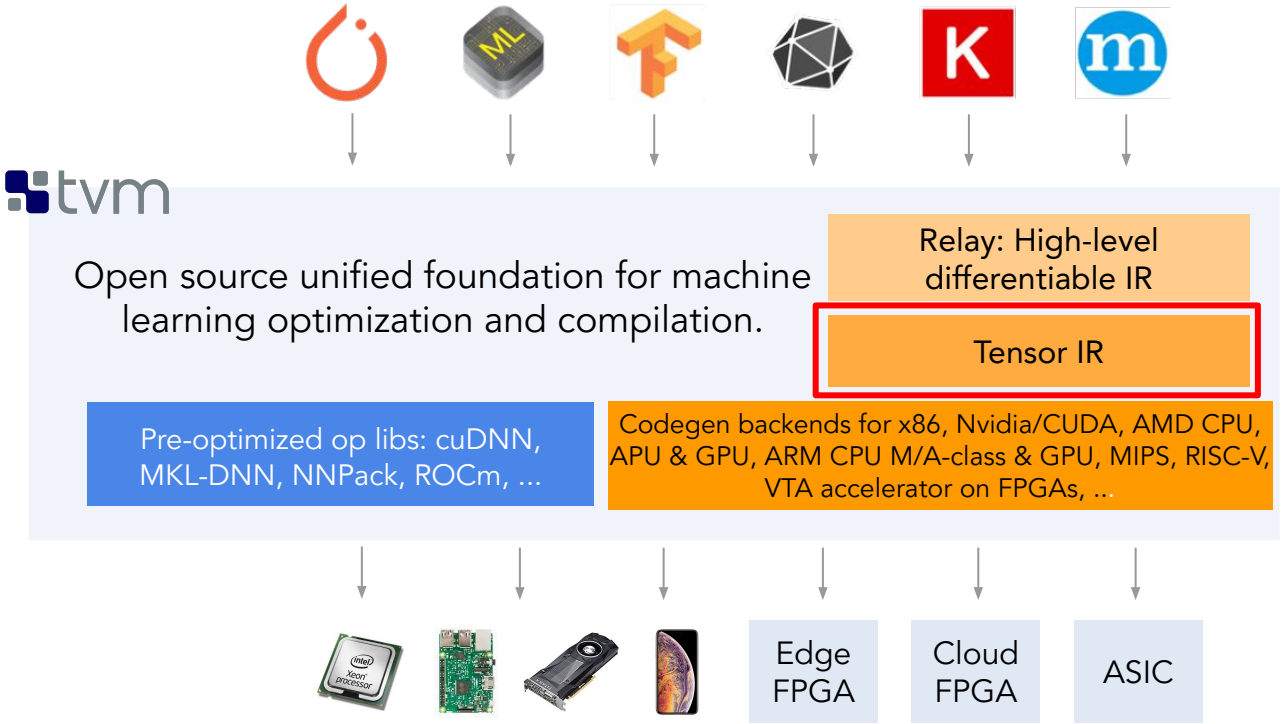
Generate efficient code for different data layouts.

Relay: Layout Transformation



Generate efficient code for different data layouts.

TVM deep dive



Low level IR represents kernels which can be optimized and compiled for all supported platforms.

Tensor IR: The Kernel Enabler

```
@tvm.script.tir
class Module:
    def mmult(A: ty.handle, B: ty.handle, C: ty.handle) -> None:
        # function attr dict
        tir.func_attr({"global_symbol": "mmult", "tir.noalias": True})
        A_1 = tir.buffer_bind(A, [1024, 1024], ...)
        B_1 = tir.buffer_bind(B, [1024, 1024], ...)
        C_1 = tir.buffer_bind(C, [1024, 1024], ...)
        # body
        tir.attr(C_1, "realize_scope", "")
        tir.realize(C_1[0:1024, 0:1024])
        for x in tir.range(0, 1024):
            for y in tir.range(0, 1024):
                C_1[x, y] = tir.float32(0)
                for k in tir.range(0, 1024):
                    C_1[x, y] = (C_1[x, y] + (A_1[x, k]*B_1[k, y]))
```

- ➔ An imperative IR for describing loop nests and low level code.
- ➔ Can represent loop-y computations, perform allocation, bind tensors to backing buffers, and so on.
- ➔ Platform agnostic CUDA-like IR.
- ➔ Provides multiple target support via LLVM, and source code generation.
- ➔ Simple cross-platform runtime API which is implemented for CUDA, Vulkan, OpenCL, Metal and so on.

Automating Kernel Code Generation is Key

AutoTVM

- Heavily template-based
- Tensorization (limited)
- Small design space
- Customize design space
- Slower compilation

AutoScheduling (e.g., Ansor)

- Template-free
- No tensorization
- Large design space
- Uncustomizable
- Faster compilation

- Automatic tensorization?
- Perf on quantized model?
- Training (AutoDiff)?
- Dynamic shape?
- Faster tuning?

OctoML

By Junru Shao, Tianqi Chen and team

Auto Tensorization

- Tensor intrinsics are important
  - NVIDIA Tensor Core
  - Intel VNNI
  - ARM `dot`
  - …
- Tensorization is hard in Ansor
  - Handling structural matching / rewriting?
- Auto Tensorization in meta schedule
  - Just a search rule!

Meta Schedule: Auto Tensorization in 4 Steps!

- Step 1: Describe the Tensor Intrinsic!

```
@tvm.script.tir
def tensorcore_desc(a: ty.handle, b: ty.handle, c: ty.handle):
  ...
  for i, j, k in tir.grid(16, 16, 16):
    ...
    C[vii, vjj] = C[vii, vjj] + A[vii, vkk] * B[vjj, vkk]

@tvm.script.tir
def tensorcore_impl(a: ty.handle, b: ty.handle, c: ty.handle):
  ...
  tir.evaluate(tir.tvm_mma_sync(
    C.data, C.elem_offset // 256,
    A.data, A.elem_offset // 256,
    B.data, B.elem_offset // 256,
    C.data, C.elem_offset // 256,
    dtype="handle",
  ))
```

Meta Schedule: Auto Tensorization in 4 Steps!

- Step 2: Automatic structural fuzzy match

```
@tvm.script.tir
def batch_matmul(a: ty.handle, b: ty.handle, c: ty.handle):
    ...
    for n, i, j, k in tir.grid(16, 128, 128, 128):
        ...
        C[vn, vi, vj] += A[vn, vi, vk] * B[vn, vj, vk]

@tvm.script.tir
def tensorcore_desc(a: ty.handle, b: ty.handle, c: ty.handle):
  ...
  for i, j, k in tir.grid(16, 16, 16):
    ...
    C[vii, vjj] = C[vii, vjj] + A[vii, vkk] * B[vjj, vkk]
```

Meta Schedule: Auto Tensorization in 4 Steps!

- Step 3: Automatic loop re-structuring & Mark tensorize region

```
@tvm.script.tir
def batch_matmul(a: ty.handle, b: ty.handle, c: ty.handle):
    ...
    for i0, i1_o, i2_o, i3_o in tir.grid(16, 8, 8, 8):
        ...
        for i1_i, i2_i, i3_i in tir.grid(16, 16, 16):
            ...
            C[vn_1, vi_1, vj_1] += A[vn_1, vi_1, vk_1] * B[vn_1, vj_1, vk_1]
```

```
@tvm.script.tir
def tensorcore_desc(a: ty.handle, b: ty.handle, c: ty.handle):
  ...
  for i, j, k in tir.grid(16, 16, 16):
    ...
    C[vii, vjj] = C[vii, vjj] + A[vii, vkk] * B[vjj, vkk]
```

Meta Schedule: Auto Tensorization in 4 Steps!

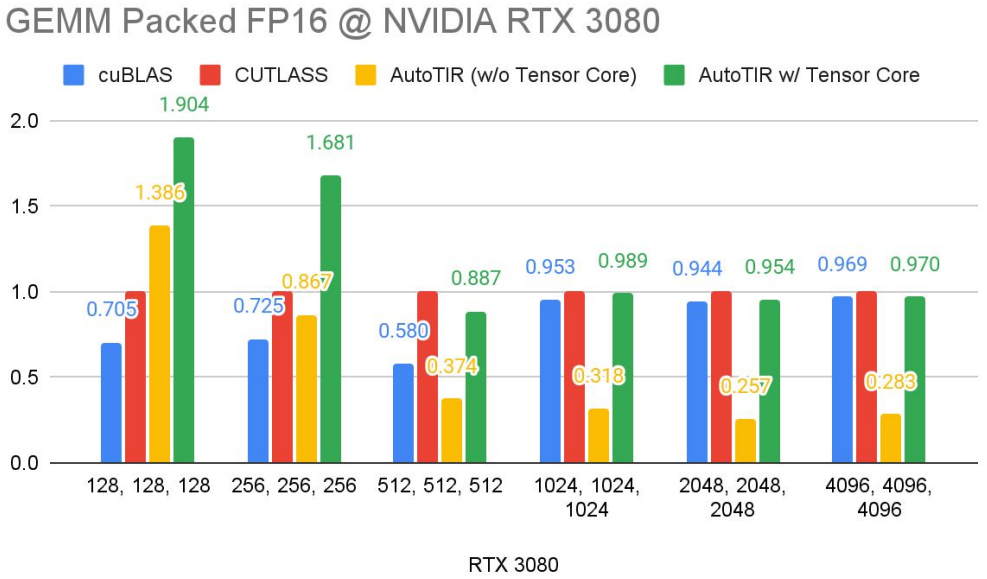
- Step 4: Working with other automatic rules + automatic tensorization

```
@tvm.script.tir
def batch_matmul(a: ty.handle, b: ty.handle, c: ty.handle):
    ...
    for i0, i1_o, i2_o, i3_o in tir.grid(16, 8, 8, 8):
        ...
        for i1_i_init, i2_i_init in tir.grid(16, 16):
            C[vn_init, vi_init, vj_init] = tir.float32(0)
        ...
        tir.evaluate(tir.tvm_mma_sync(
            C.data, tir.floordiv(tir.get_elem_offset(C[vn_1, vi_1, vj_1], dtype="int32"), 256),
            A.data, tir.floordiv(tir.get_elem_offset(A[vn_1, vi_1, vk_1], dtype="int32"), 256),
            B.data, tir.floordiv(tir.get_elem_offset(B[vn_1, vj_1, vk_1], dtype="int32"), 256),
            C.data, tir.floordiv(tir.get_elem_offset(C[vn_1, vi_1, vj_1], dtype="int32"), 256),
            dtype="handle",
        ))
```

….and it looks very promising!

(by Bohan from CMU)

- Surpasses cuBLAS perf
- Matches CUTLASS (tuned) perf
- 1.9x on small shapes
- 95% on large shapes
- Tensorized



Best of both worlds

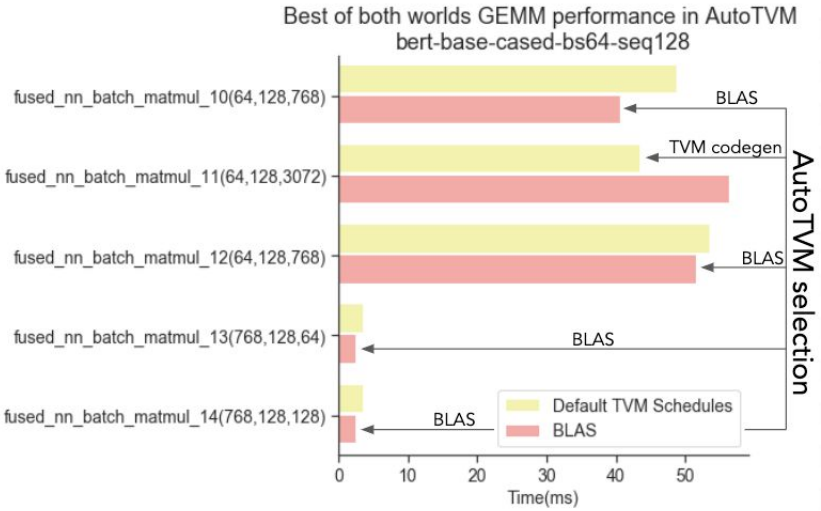
We care about performance, coverage, and portability, not code generation ideology.

{cuDNN, MKL, ARM Compute Lib, TensorRT, …}

Best of both worlds

```
For each kernel (or supported subgraph):
     Use argmax(code generation, kernel library)
```

Results: up to 40% gains over TensorRT on Nvidia T4



Select Performance Results

Performance at OctoML in 2020/2021

TVM $\log_2$ fold improvement over baseline

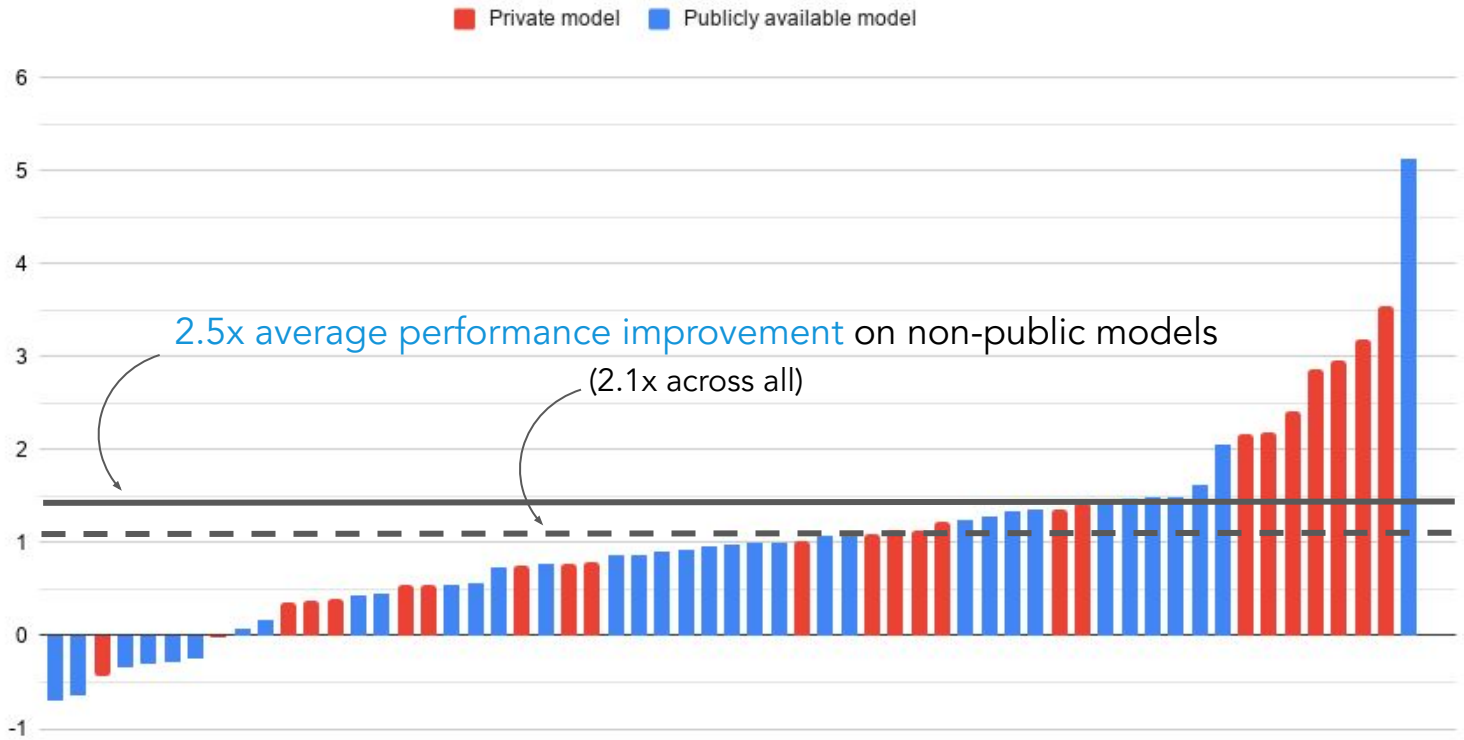
Over 60 model x hardware benchmarking studies

Each study compared TVM against best* baseline on the target

Sorted by ascending $\log_2$ gain over baseline

Model x hardware comparison points

Higher performance on non-public models

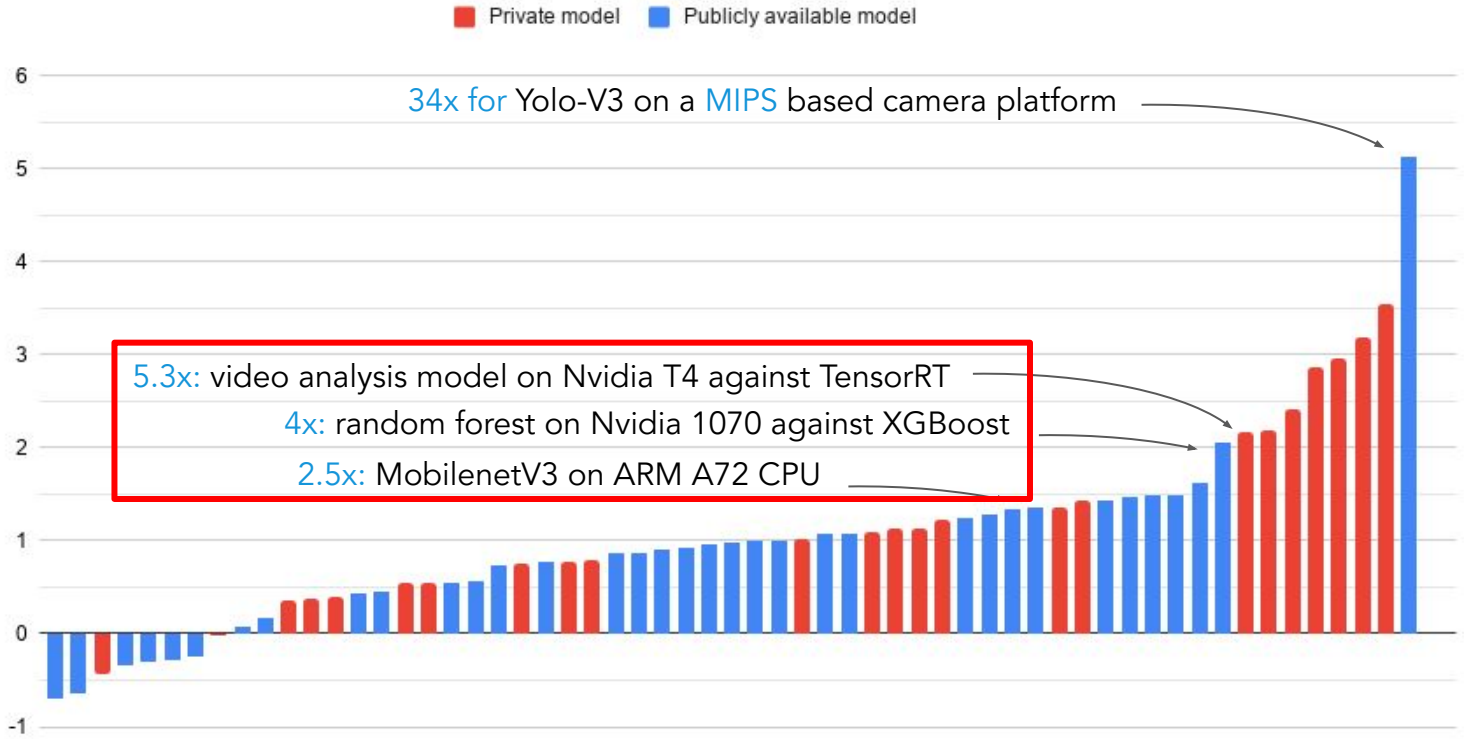


TVM $\log_2$ fold improvement over baseline

Model x hardware comparison points

OctoML

Across a broad variety of models and platforms



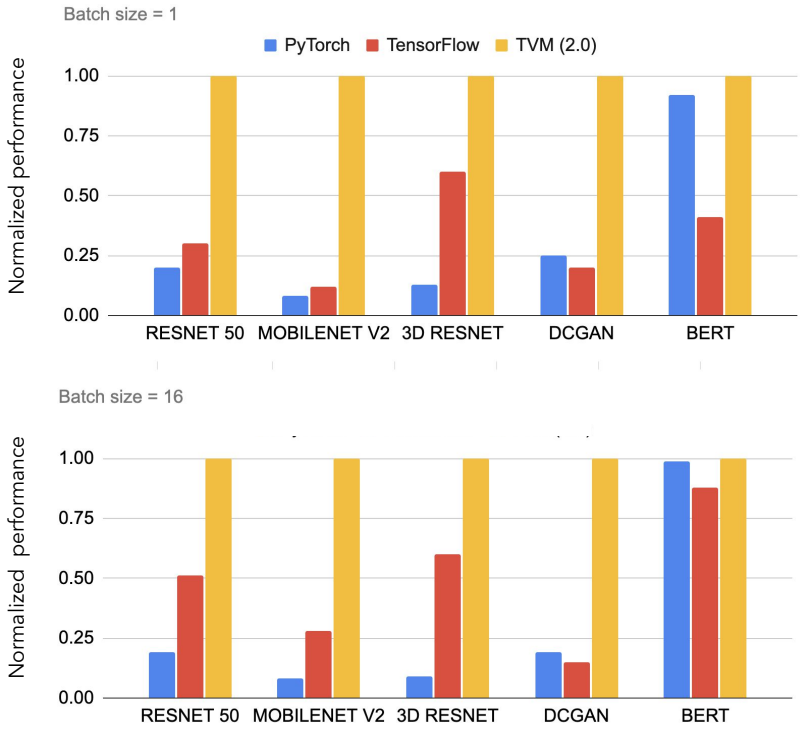
TVM $\log_2$ fold improvement over baseline

Model x hardware comparison points

OctoML

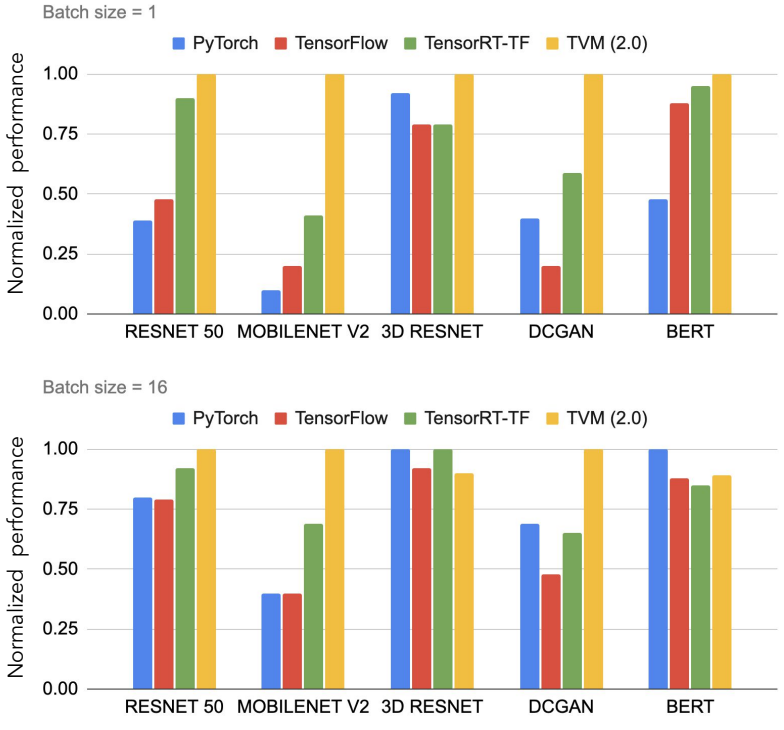
Results: TVM on popular CPUs and GPUs

Intel X86 - 2-5X Performance



20 core Intel-Platinum-8269CY fp32 performance data

NVIDIA GPU - 20-50% versus TensorRT



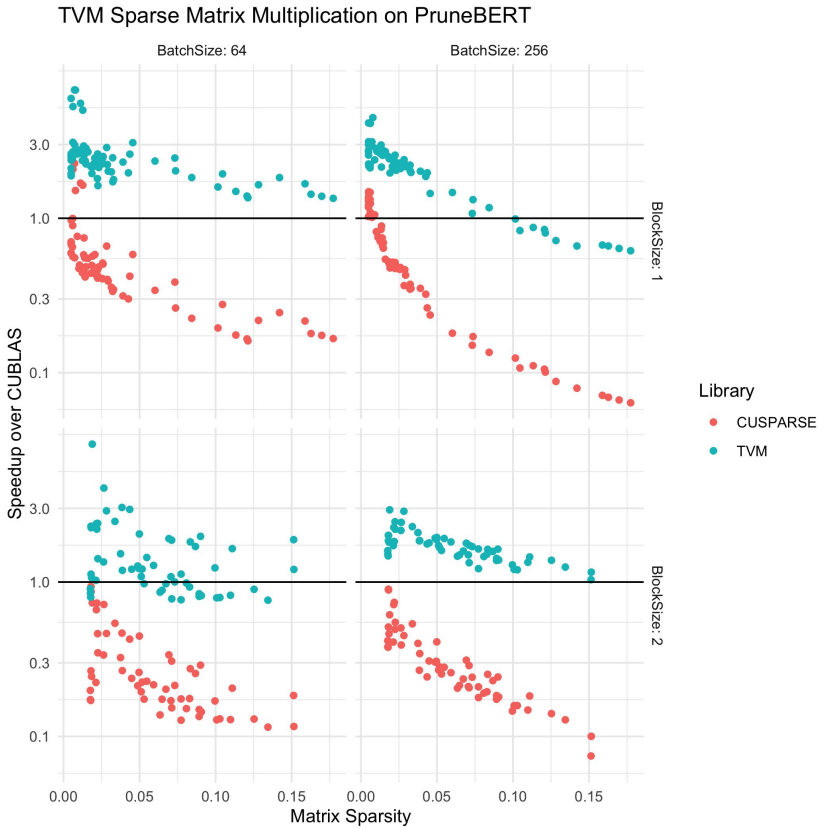
V100 fp32 performance data

OctoML

See https://github.com/tlc-pack/tlcbench for benchmark scripts

Faster Kernels for Dense-Sparse Multiplication

- Performance comparison on PruneBERT
- 3-10x faster than cuBLAS and cuSPARSE.
- 1 engineer writing TensorIR kernels



OctoML

Classical ML (Hummingbird)

https://github.com/microsoft/hummingbird

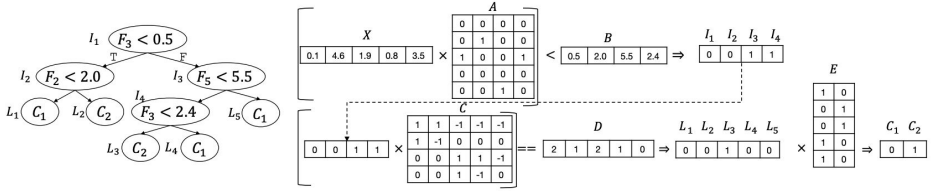
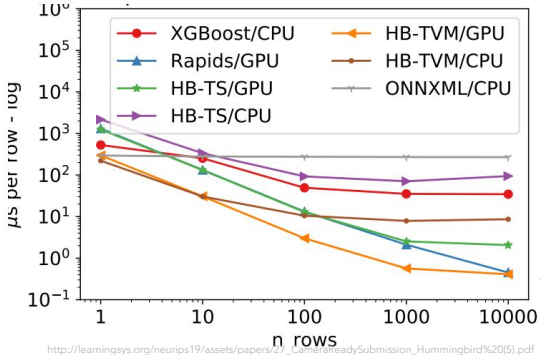


Figure 3: Compiling an example decision tree using the GEMM strategy (algorithm 1).



http://learningsys.org/neurips19/assets/papers/27_CameraReadySubmission_Hummingbird%20(5).pdf

Tree-models Microbenchmark: Batch w/ GPU

	rf	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	2.52s	8.1s	0.15s	0.11s	17.55s	**0.02s**
year	2.33s	17.23s	0.15s	0.10s	45.12s	**0.03s**
covtype	47.64s	24.77s	0.32s	0.26s	62.61s	**0.06s**
epsilon	11.22s	26.03s	0.36s	0.28s	OOM	**0.14s**

60x

	xgb	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	2.01s	6.4s	0.16s	0.11s	16.89s	**0.02s**
year	5.77s	15.75s	0.14s	0.1s	44.82s	**0.03s**
covtype	63.45s	173.92s	1.47s	1.29s	445.89s	**0.25s**
epsilon	14.84s	29s	0.37s	0.28s	OOM	**0.14s**

200x

	lgbm	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	3.76s	6.59s	0.16s	0.11s	17.70s	**0.02s**
year	6.18s	10.14s	0.14s	0.10s	OOM	**0.03s**
covtype	67.12s	158.3s	1.47s	1.29s	446s	**0.25s**
epsilon	14.13s	26.03s	0.36s	0.28s	OOM	**0.14s**

https://sampl.cs.washington.edu/tvmconf/slides/2019/E13-Matteo-Interlandi.pdf

Tree-models Microbenchmark: Batch Inference on CPU

	rf	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	**2.52s**	8.1s	17.18s	17.28	92.58s	3.84s
year	2.33s	17.23s	17.95s	17.23s	154.71s	**1.43s**
covtype	47.64s	24.77s	38.27s	38.02s	260.35s	**20.18s**
epsilon	11.22s	26.03s	48.52s	45.87s	SEGFAULT	**8.17s**

50%

	xgb	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	2.01s	6.4s	17.23s	16.38s	89.71s	**1.93s**
year	5.77s	15.75s	17.26s	15.74s	153.96s	**1.77s**
covtype	63.45s	173.92s	295.6s	295.3s	1255s	**28.53s**
epsilon	14.84s	29s	47.38s	48.78s	SEGFAULT	**4.43s**

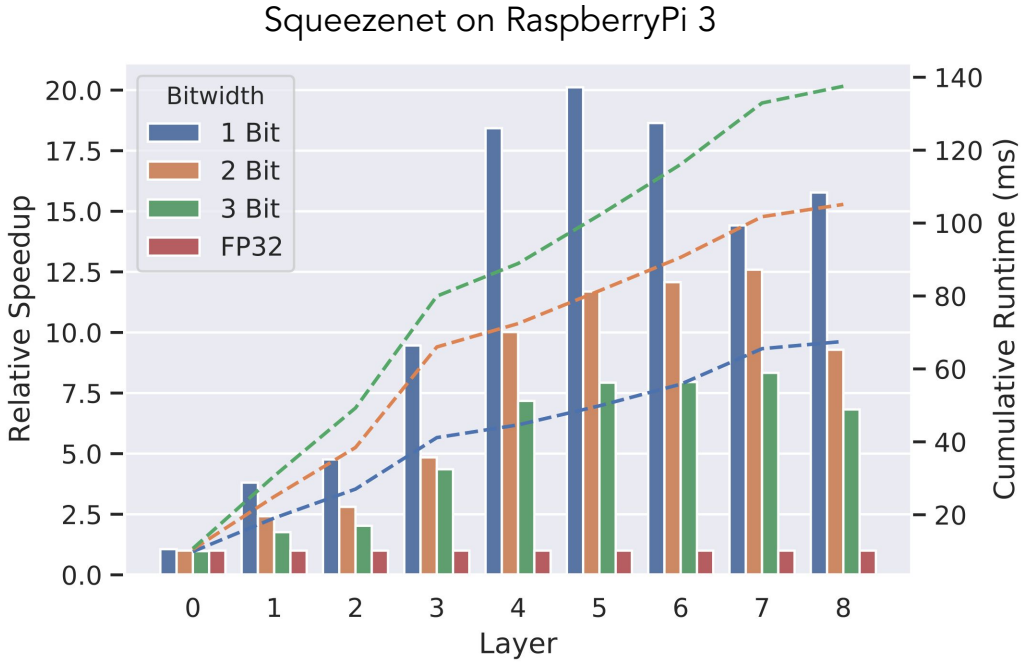
	lgbm	onnx-ml	hb-pt	hb-ts	hb-onnx	hb-tvm
fraud	3.76s	6.59s	17.41s	16.43s	89.90s	**1.97s**
year	6.18s	10.14s	18.3s	18.01s	153.67s	**1.78s**
covtype	67.12s	158.3s	296s	294s	1256s	**29.19s**
epsilon	14.13s	26.03s	47.21s	47.89s	SEGFAULT	**4.41s**

3x

https://sampl.cs.washington.edu/tvmconf/slides/2019/E13-Matteo-Interlandi.pdf

Ultra low bit-width quantization

- In addition to fp32, fp16, int8
- TVM supports bitserial ultra low bit code generation
  - Int{4,3,2,1}
- See Josh Fromm's MLSys 2020 talk and paper.



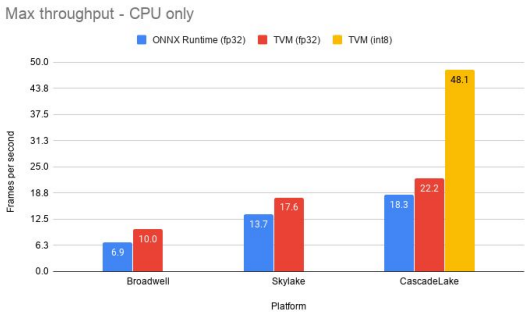
Case Study: 90% cloud inference cost reduction

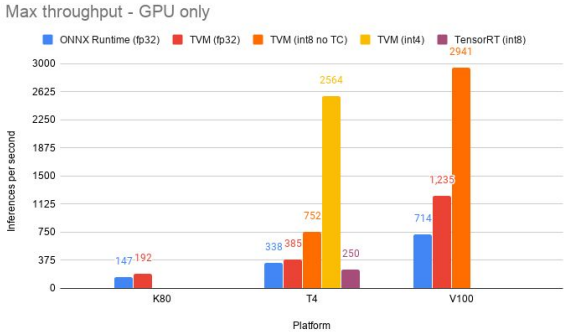
Background

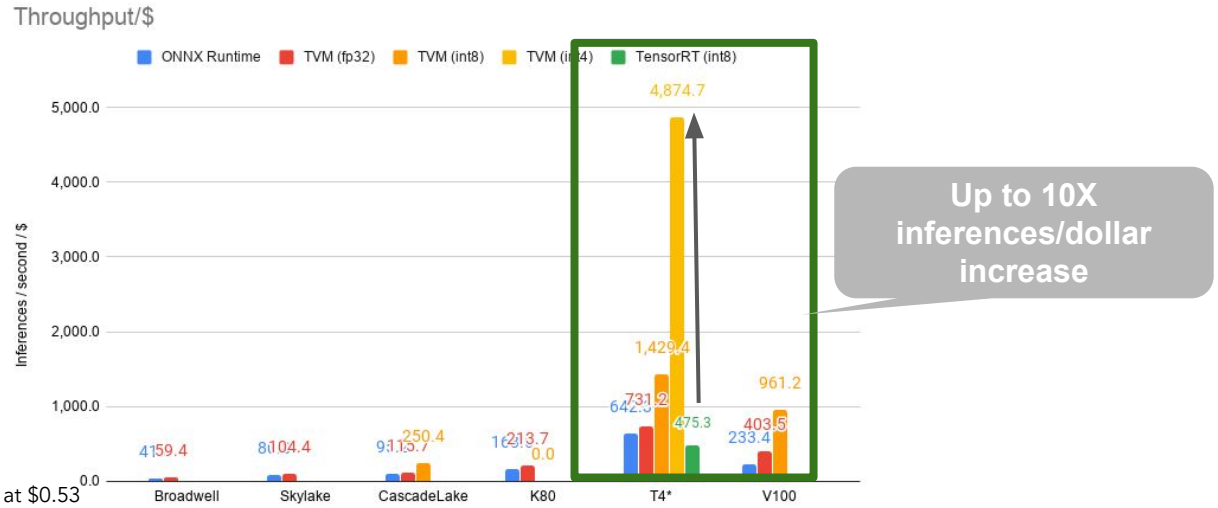
- Top 10 Tech Company running multiple variations of customized CV models
- Model in batch processing /offline mode using standard HW targets of a major public cloud.
- Billions of inferences per month
- Benchmarking on CPU and GPU

Results

- 3.8x - TensorRT 8bit to TVM 8bit
- 10x - TensorRT 8bit to TVM 4bit
- **Potential to reduce hourly costs by 90%**



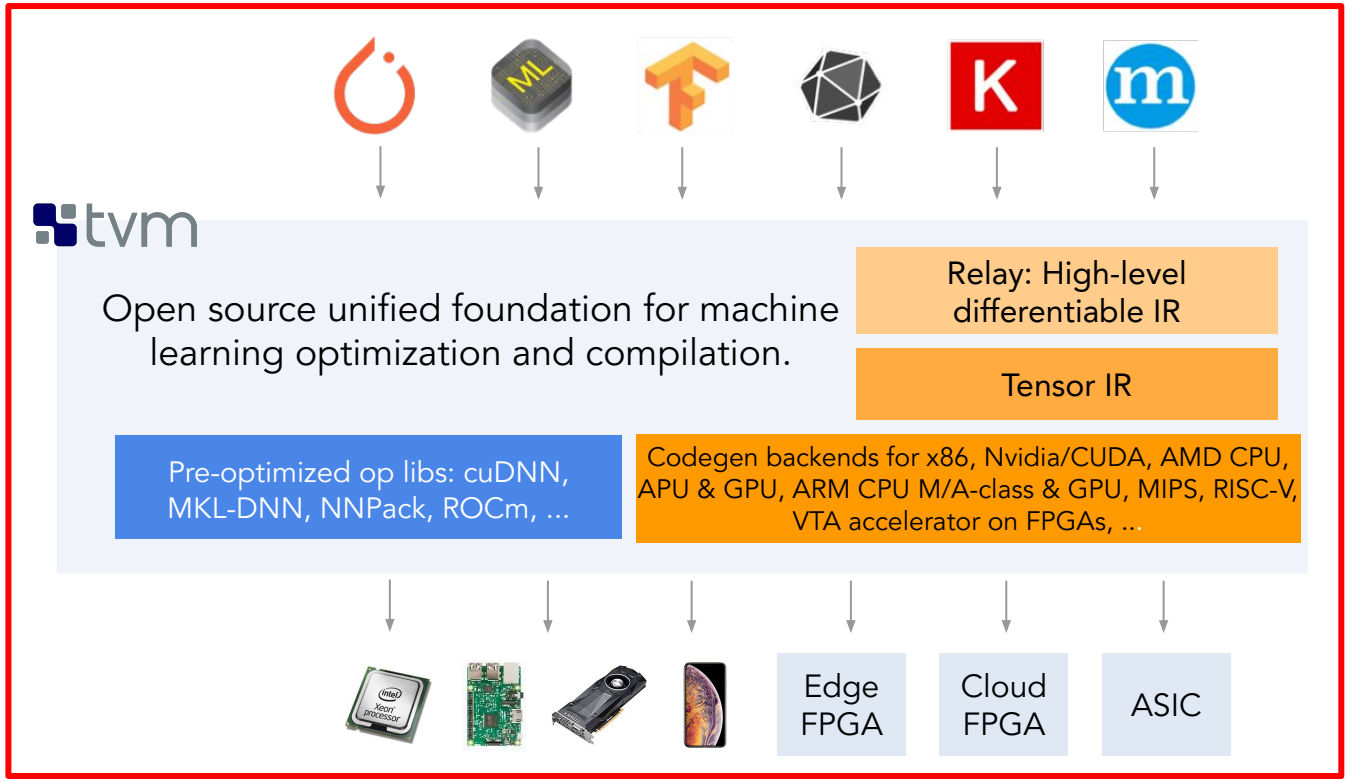




*V100 at hourly price of $3.00 per hour, T4 at $0.53

OctoML

Making the most out of TVM & other stacks...



How do end users choose between all the possible configurations of:

- Model definitions
- Optimizations
- Kernel Combinations
- Target Devices
- And more

Automated platform for ML acceleration & deployment

Models: TensorFlow, PyTorch, ONNX, ...

OctoML platform

Hosted service: No user infrastructure needed.

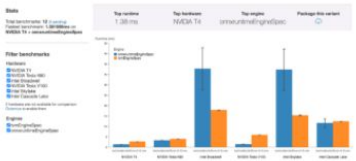
Tuned optimization: Leveraging OctoML's ML tuning data.

Automated/flexible packaging: Easy integration.

Optimized Model

- Device-specific tuning for each HW platform
- Leverages tuning data from similar models, HW

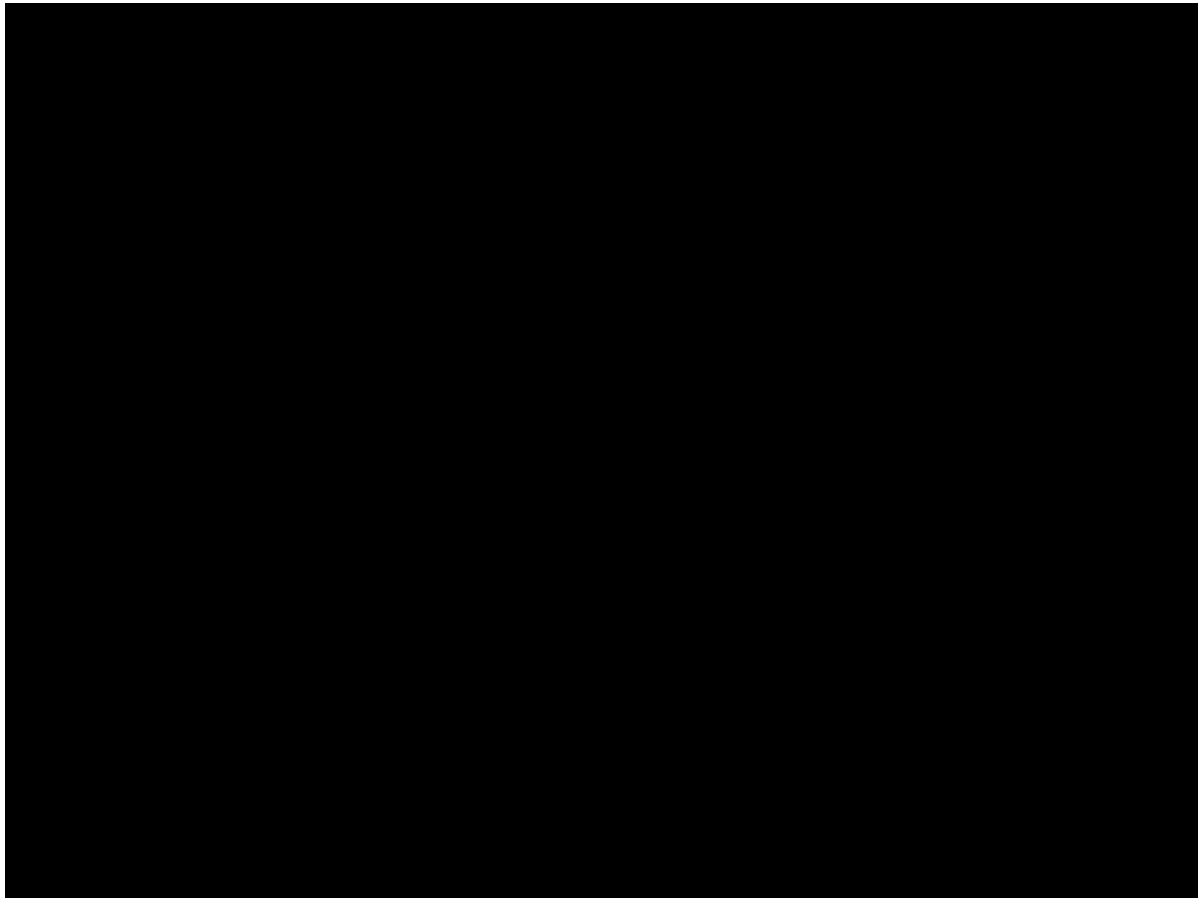
Benchmarking Data



Packaging

- TVM C Runtime and C API
- Python API
- gRPC
- Docker

Octomizer Demo



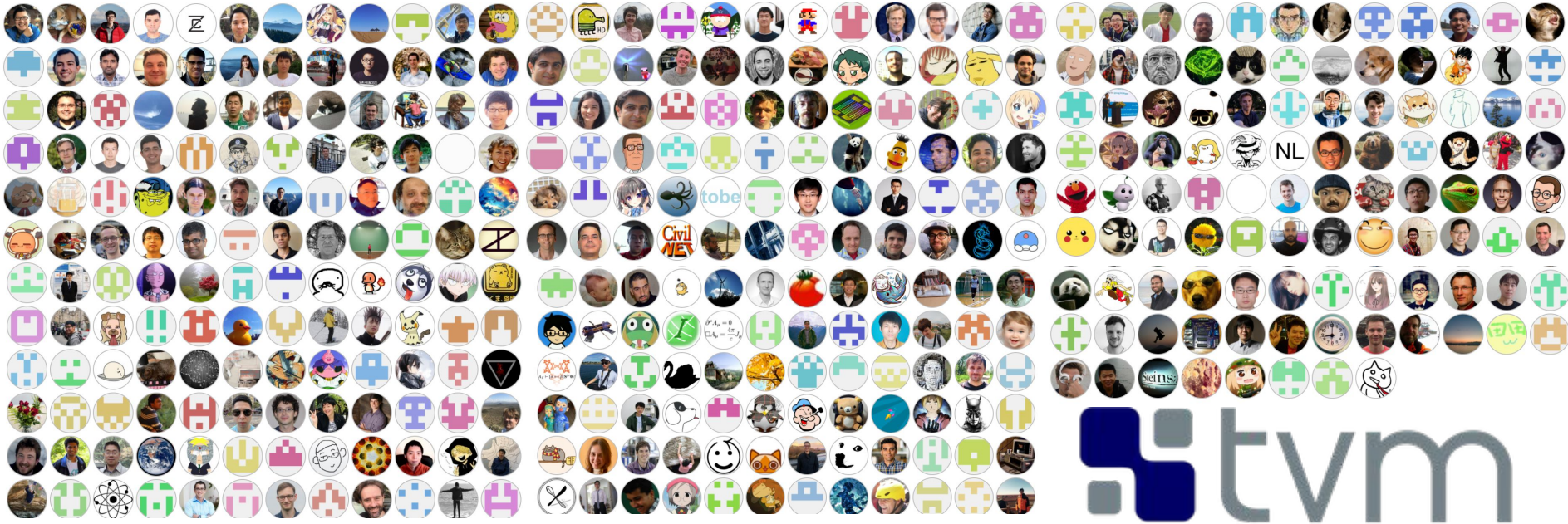
API access

```
# Specify model file and input layer parameters.
model_file = "mnist.onnx"
input_shapes = {"Input3": (1, 3, 28, 28)}
input_dtypes = {"Input3": "float32"}
model = onnx_model.ONNXModel(client, input_shapes, input_dtypes)

# Upload the file to Octomizer.
modelvar = model.upload("mnist.onnx")

# Octomize it.
wrkflow = modelvar.octomize(platform="broadwell")
wrkflow.wait()
wrkflow.save_package("mnist-octomized.whl")
```

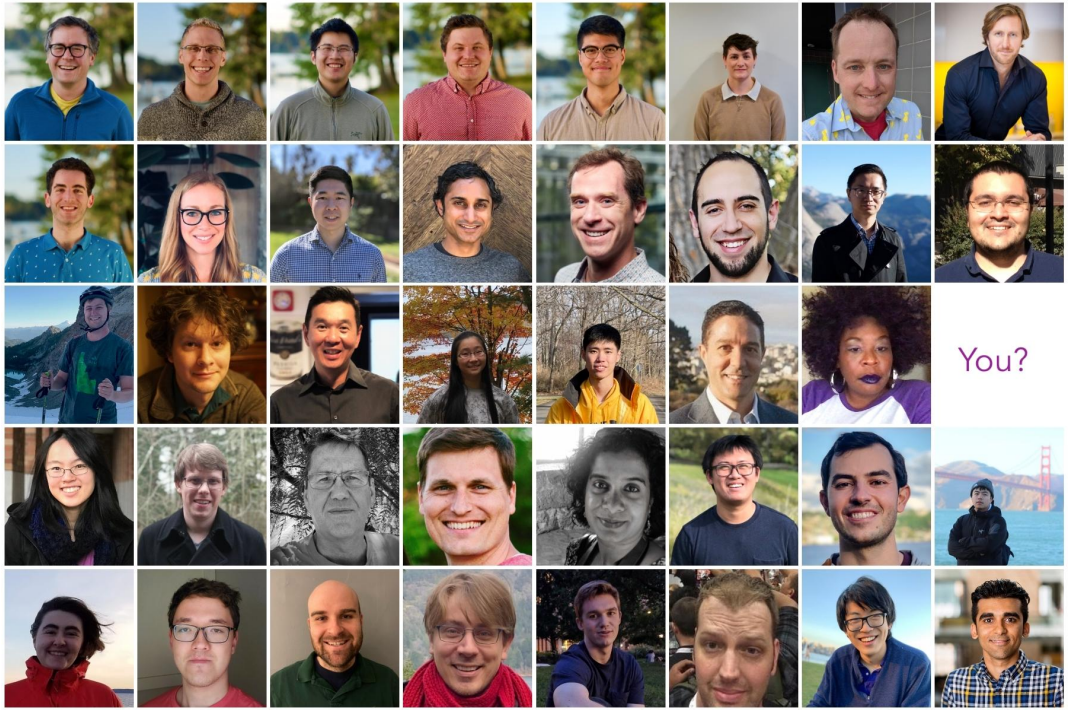
Waitlist! https://octoml.ai

TOYOTA, iRobot, AMD, Narrative, Raytheon, rebellion, verizon, XILINX, CREDIT SUISSE, Texas Instruments, selcom, BOSCH, Honeywell, citi, LG, AB InBev, SYNOPSYS

Thank you Apache TVM community! 600+!



The Octonauts!



You?

https://octoml.ai/careers

Thank you!

Improving Model Performance, Portability and Productivity with Apache TVM and the OctoML platform

ModSim'21

Luis Ceze

Co-founder & CEO, OctoML.

Professor, University of Washington.

OctoML

OctoML

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

TVM, ONNX-Runtime, and MLIR